Université Félix Houphouët-Boigny
(UFHB)

Cours d'algorithme avancée
L3-Info

Enseignant :

Dr GBAME Gbede Sylvain

Assistant, enseignant chercheur à l'UFHB

gbamegbedesylvain@gmail.com

MODULE : Algorithme avancée

PLAN

Chapitre 7
Les Structures de données

7.0 Introduction

1.1 Objectif du cours

L'objectif du cours est de présenter les structures de données et les algorithmes permettant de les manipuler. Nous aurons une introduction générale aux structures de données. Ensuite, nous allons fournir les outils nécessaires pour l'organisation et la manipulation des données dans les structures de données. Enfin, nous allons aborder quelques algorithmes avancés et étudier leur complexité.

1.2 Généralités des SD

1.2.1 Rappel sur le rôle des ordinateurs

Un ordinateur est un appareil ou une machine qui permet de réaliser, d'exécuter des opérations, des calculs, des tâches. De ce fait une simple calculatrice est un ordinateur (figure 1). Un ordinateur n'est pas à lui seul capable de résoudre des problèmes. Ce sont les algorithmes qui sont chargés de résoudre ou faire résoudre les problèmes aux ordinateurs en leur donnant les taches à exécuter

7.0 Introduction

1.2.2 Définition Algo

Un algorithme est la description d'une suite d'étapes permettant d'obtenir un résultat à partir d'éléments fournis en entrée. Pour qu'un algorithme puisse être exécuté par un ordinateur, il faut le transformer en langage machine (compilation).

Figure 1 : Calculatrice

7.0 Introduction

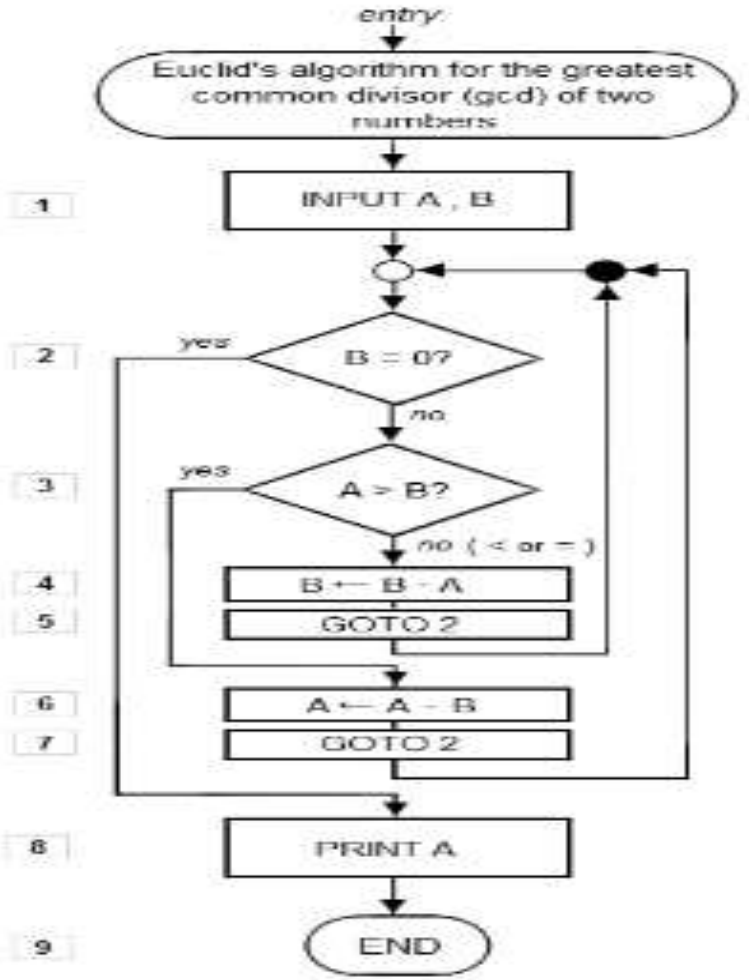

Figure 2 : Représentation d'un Algorithme

7.0 Introduction

Un algorithme reçoit des données en entrée qu'il manipule pour produire un résultat à la sortie en ayant effectué des opérations (figure 2). De cette phrase, nous retenons deux choses : **manipulation** des données et **exécution** des opérations. Dans ce cours nous allons nous intéresser à la manipulation des données.

Pour manipuler les données tout au long de la réalisation (le temps d'exécution) de l'algorithme, elles doivent être stockées dans la mémoire vive de l'ordinateur (RAM). De plus, les données manipuler peuvent varier. On parle alors de **variable**.

7.0 Introduction

1.2.3 Variables (SD élémentaire)

Dans un programme informatique, on va avoir en permanence besoin de stocker provisoirement des valeurs. Il peut s'agir de données issues du disque dur, fournies par l'utilisateur (entrées au clavier). Il peut également s'agir de résultats obtenus par le programme, intermédiaires ou définitifs. Ainsi, au sein du même algorithme, les valeurs d'une variable sont amenées à changer. Les données des variables peuvent être de plusieurs types : elles peuvent être des nombres, du texte, des dates, des images, etc.

Une variable est une boîte ou case de la mémoire, que le programme réserve et repérée par un nom. Pour avoir accès au contenu de la boîte, il suffit de la désigner par son nom. Dans la réalité, la case mémoire est repérée par une adresse. C'est dans le programme que la variable porte un nom. C'est l'ordinateur qui fait le lien entre le nom et l'adresse lors de la compilation.

7.0 Introduction

Une variable est donc identifiée par :

- Un nom
- Un type
- Une valeur
- Une adresse (facultative car c'est l'ordinateur qui s'en occupe)

Les types de données les plus couramment utilisés sont,
les *entiers*, les *réels*, les *chaines de caractère*, les *booléen*, etc. la mémoire réservée dépend du type de la variable (tableau 1).

Chapitre 7 : Les Structures de données

7.0 Introduction

Tableau 1 : Type de variable en algorithme

Type	Codage	Valeurs
Caractère	2 octets	'a', 'A', "", ' '
Entier	4 octets	2147418112
Réel	8 octets	134.456, -45E-16
Booléen	1 octet	False, True
Chaîne de caractères	Selon le nombre de caractères	"INF1563", "M1 Math"
Date	8 octets	05/04/2022

Ces types de variables sont suffisant pour manipuler les données de base. Cependant, lorsqu'il faut manipuler des données complexes et structurées, il faut des types de variables plus évolués. C'est à ce niveau qu'intervient les structures de données

7.0 Introduction

1.2.4 Définition des SD

La plupart des bons algorithmes fonctionnent grâce à une organisation intelligente des données. Une structure de données est une représentation structurée des données. Elle permet de mieux organiser et stocker les données qui sont généralement liées. Le but est de faciliter l'accès et la manipulation. Par exemple, les noms des étudiants peuvent être stocker dans la structure de données "Tableau".

1.2.5 Importance des SD

Les structures de données sont essentielles pour gérer efficacement de grandes quantités de données, telles que les informations conservées dans des bases de données ou des services d'indexation. La maintenance correcte des systèmes de données nécessite l'identification de l'allocation de la mémoire, des interrelations entre les données et des processus de données, autant d'éléments auxquels les structures de données contribuent.

7.0 Introduction

En outre, il est non seulement important d'utiliser des structures de données, mais aussi de choisir la structure de données appropriée pour chaque tâche. Le choix d'une structure de données inadaptée pourrait entraîner des temps d'exécution lents ou un code non réactif.
Quelques facteurs à prendre en compte lors du choix d'une structure de données incluent le type d'informations qui seront stockées, l'emplacement des données existantes, la manière dont les données doivent être triées et la quantité de mémoire à réserver pour les données.

Pour gérer le milliard de pages web indexées par google ou les informations des habitants d'un pays comme la chine (1,3 milliard d'habitants) les systèmes informatiques doivent prévoir la taille de la mémoire à allouer et la disposition des données. Les types de variables élémentaires sont inefficace pour de tels traitements. Par exemple, prenons la gestion d'une cérémonie avec 200 invités. Connaitre le nombre n'est pas suffisant, vous devez avoir des informations sur le type d'invités, est-ce une cérémonie pour adulte ou pour enfant ? (Prendre l'analogie des tréteaux) y a-t-il des invités de marque ? etc. voilà autant d'information pour prévoir la disposition des chaises et tables, des couverts, etc

7.0 Introduction

1.2.6 Usage des SD

Les structures de données contiennent des informations sur des valeurs, des données, des relations entre les données et les fonctions qui peuvent être appliquées aux données. Il existe de nombreuses structures de données qui peuvent être utilisées dans des domaines variés. Cependant les structures de données qui seront étudiées dans ce cours sont les suivants :

Enregistrement

La Structure de données Enregistrement sert à rassembler au même endroit les informations sur une même entité. Par exemple, les informations sur les étudiants de l'université sont organisées dans une variable de type Enregistrement.

7.0 Introduction

Pile et File

La pile permet de stocker des éléments et les récupérer dans l'ordre inverse. L'élément qui est stocké à la fin sera le premier à être récupéré de la pile.

Un éditeur de texte tel que Notepad ou Microsoft Word utilise une structure de données Pile pour accomplir des tâches d'annulation dans son éditeur. Un autre bon exemple de pile est la gestion de l'historique dans le navigateur de votre ordinateur portable ou de votre système.

Liste Chaînée

Les listes chainées stockent des collections d'éléments dans un ordre linéaire où chaque élément est lié au suivant.

7.0 Introduction

Exemple, lorsqu'on crée une liste de lecture sur un smartphone, elle fonctionne sur le concept de la liste chainée. Ces chansons sont jouées une par une, elles sont connectées et on peut passer de la chanson trois à la chanson quatre.

Tableau

La première structure de données à apprendre en programmation et la structure de données la plus courante et la plus utilisée dans de nombreuses applications.

Pointeur

Le pointeur n'est pas à proprement dit une structure de données. Il constitue plutôt une référence vers une zone mémoire. Il est utilisé pour la gestion dynamique de la mémoire.

Arbre

Un arbre stocke une collection d'éléments de manière abstraite et hiérarchique. Chaque nœud est lié à d'autres nœuds et peut avoir plusieurs sous-valeurs, également appelées enfants. Il est utilisé en intelligence artificielle et data analyses.

7.1 Enregistrement

Supposons que nous voulons enregistrer les informations des étudiants de M1 Maths. Les informations qui nous intéressent sont : le *nom*, le *prénom*, la *date de naissance*, la *taille*, *boursier* ou non. Comment nous allons procéder, quelles sont les variables que nous allons utiliser ?

3.1 Définition

Les enregistrements (également nommés types structurés) sont des structures de données dont les éléments constitutifs peuvent être de type différent et qui se rapportent à la même entité. L'avantage d'un enregistrement est de regrouper des informations sémantiquement liées pour éviter de multiplier les variables et bien faire apparaitre la logique du traitement.
Jusqu'à présent, nous n'avons utilisé que des types primitifs (caractères, entiers, réels, chaînes) et des tableaux de types primitifs. Mais nous pouvons créer nos propres types puis déclarer des variables ou des tableaux d'éléments de ce type. Pour créer des enregistrements, il faut déclarer un nouveau type, basé sur d'autres types existants.

7.1 Enregistrement

3.2 Déclaration

On déclare une structure de données de la manière suivantes :

TYPE
Nom_enregistrement = ***Enregistrement***
Champ1 : ***type1*** *;*
Champ2 : ***type2*** *;*
Fin Enregistrement

7.1 Enregistrement

Prenons l'exemple de l'enregistrement des informations d'un étudiant de M1 Maths. Nous aurons :

```
TYPE
Tetudiant = Enregistrement
    Nom : Chaine de caractère ;
    Prénom : Chaine de caractère ;
    Date_Naissance : Date ;
    Taille : Réel ou entier ;
    Boursier : Booléen ;
Fin Enregistrement
```

Une fois qu'on a défini un enregistrement, on peut déclarer des variables enregistrements exactement de la même façon que l'on déclare des variables d'un type primitif.

```
Variables
    Nom_variable : nom_enregistrement ;
```

7.1 Enregistrement

Exemple

Variables

Etudiant1, Etudiant2 : Tetudiant ;

3.3 Manipulation des enregistrements

La manipulation d'un enregistrement se fait au travers de ses champs. Comme pour les tableaux, il n'est pas possible de manipuler un enregistrement globalement, sauf pour affecter un enregistrement à un autre de même type. Par exemple, pour afficher un enregistrement il faut afficher tous ses champs uns par uns. Il n'est donc pas possible de faire ;

~~*Afficher (Etudiant1)*~~

Pour avoir accès à un champs d'un enregistrement, il faut mettre un "." entre le nom de l'enregistrement et le nom du champ. Par exemple :

Etudiant1.Nom

7.1 Enregistrement

3.4 Tableau d'enregistrement

Supposons que nous ayons maintenant 500 étudiants que nous devons enregistrer. Il serait fastidieux de créer 500 variables de type Tetudiant. On va créer un tableau regroupant toutes les personnes du groupe. Il s'agit alors d'un tableau d'enregistrements.

Variables

Nom_variable : tableau[1..N] de nom_enregistrement ;

Exemple

Variables

Etudiant : tableau[1..500] de Tetudiant ;

7.1 Enregistrement

Cela représente :

	Nom	Prénom	Date de naissance	Taille	Boursier
1					
2					
…					
500					

Pour accéder au nom du deuxième étudiant on fait :

Etudiant[2].Nom

et pas

~~*Etudiant.Nom[2]*~~

7.2 Les Piles

4.1 Définition

Une pile est une structure qui stocke de manière ordonnée des éléments, mais rend accessible uniquement un seul d'entre eux, appelé le sommet de la pile. Cette structure de données applique la stratégie LIFO (Last In, First Out). Le dernier élément ajouté est le premier à sortie (figure 3).

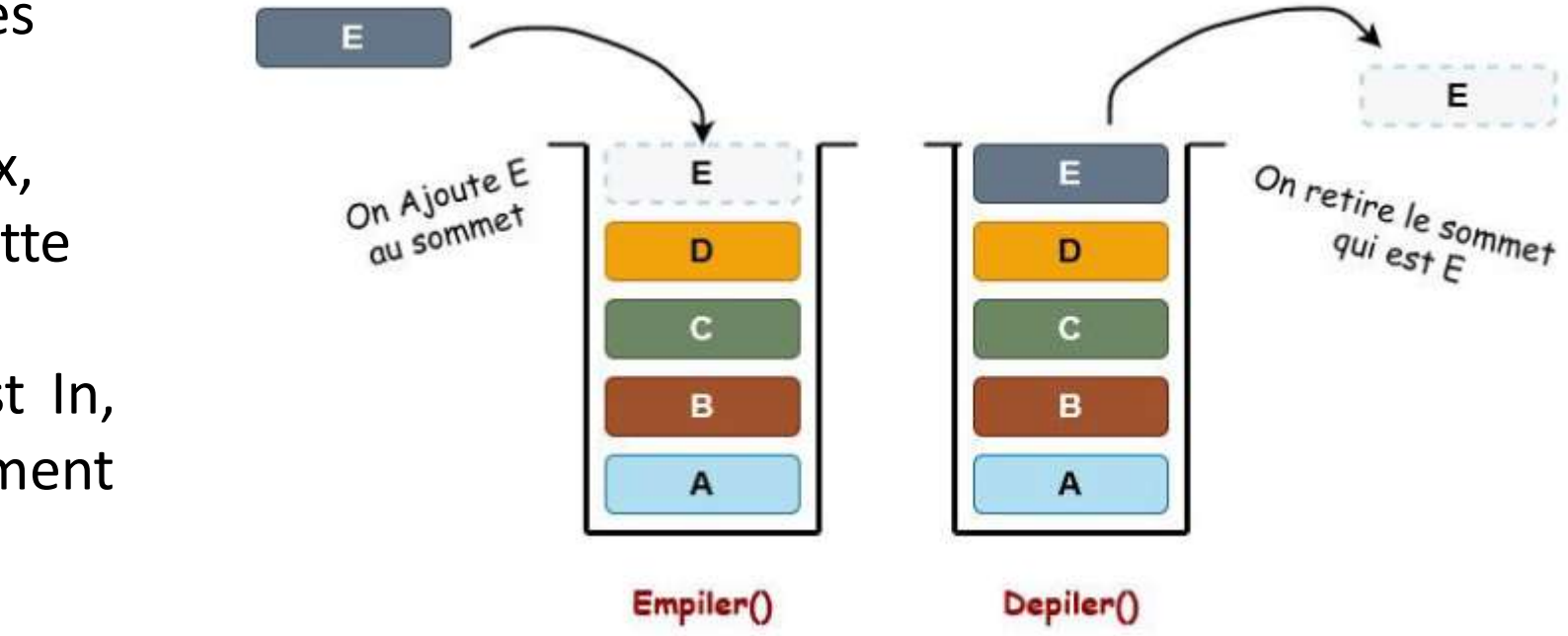

Figure 3 : Représentation des piles

7.2 Les Piles

4.2 Modélisation d'une pile

Il existe deux manières pour modéliser une pile. Soit par un tableau soit par une liste chainée. La modélisation par **tableau** consiste à utiliser un tableau de taille fixe pour insérer les éléments de la pile. L'ajout d'un élément se fera à la suite du dernier élément du tableau. Le retrait d'un élément de la pile se fera en enlevant le dernier élément du tableau. De ce fait, il est nécessaire de connaitre la position du dernier élément de la pile, C'est le **sommet** de la pile (figure 4).

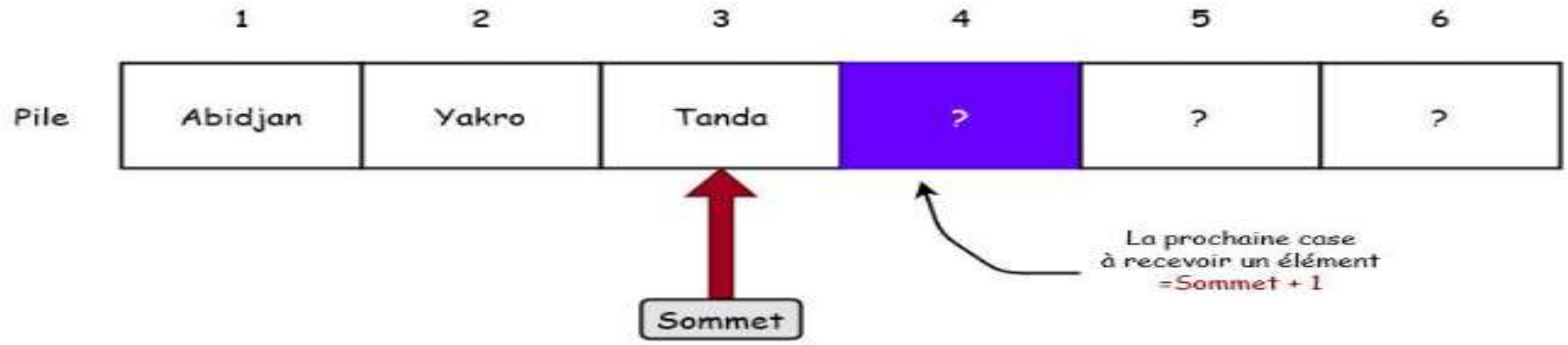

Figure 4 : Modélisation d'une pile avec le sommet

7.2 Les Piles

Les invariants ou les propriétés d'une pile sont : le *nom* de la pile, le *sommet*, le *tableau* d'élément et le *type*. Après la déclaration de la pile, les opérations qui peuvent être réalisées sur celle-ci (également appelées primitives de la pile) sont : l'initialisation, le test de pile vide, le test de pile pleine, l'empilement et le dépilement.

4.3 Déclaration et initialisation d'une pile

Pour construire une pile nous avons besoin d'un tableau de taille fixe et de l'indice du sommet. La structure de données qui peut regrouper ces deux informations de type différent est l'enregistrement.

7.2 Les Piles

4.3.1 Déclaration

```
TYPE
Nom_pile = Enregistrement
    Const entier taille = 10;
    P : tableau [taille] de entiers ;
    sommet : entier ;
Fin Enregistrement
```

4.3.2 Initialisation

A l'initialisation, le sommet de la pile est égal à zéro.

```
Initialiser (pile p)
p.sommet = 0;
Fin
```

7.2 Les Piles

4.3.3 Les primitives ou opérations des piles

Test de pile vide

Une pile est vide lorsque le sommet est égal à zéro.

```
Estvide(pile p)
    Si p.sommet = 0 alors
        Retourner vrai ;
    Sinon
        Retourner faux ;
Fin
```

Test de pile pleine

7.2 Les Piles

La pile est pleine lorsque le sommet est égal à la taille N du tableau.

```
Estpleine(pile p)
    Si p.sommet = p.taille alors
            Retourner vrai ;
    Sinon
            Retourner faux ;
Fin
```

Empilement

7.2 Les Piles

L'empilement consiste à ajouter un élément dans la pile. Ainsi, le nouvel élément se mettra au-dessus du sommet actuel et deviendra le nouveau sommet. La pile ne doit pas être pleine.

```
Empiler(pile p, x)
    Si estpleine(p) = faux alors
            p.sommet = p.sommet + 1;
            p[p.sommet] = x;
    Sinon
            Afficher ('pile déjà pleine') ;
Fin
```

7.2 Les Piles

L'empilement consiste à ajouter un élément dans la pile. Ainsi, le nouvel élément se mettra au-dessus du sommet actuel et deviendra le nouveau sommet. La pile ne doit pas être pleine.

```
Empiler(pile p, x)
    Si estpleine(p) = faux alors
            p.sommet = p.sommet + 1;
            p[p.sommet] = x;
    Sinon
            Afficher ('pile déjà pleine') ;
Fin
```

7.2 Les Piles

Dépilement

Le dépilement consiste à enlever un élément de la pile. Du fait de la stratégie LIFO, c'est le sommet qui peut être enlevé. Ainsi, en retirant un le sommet, c'est la case précédente qui devient le nouveau sommet. Cependant, il faut s'assurer que la pile n'est pas vide, sinon l'opération retournera une erreur.

```
Depiler(pile p)
    x : entier ;
    Si estvide(p) = faux alors
            x = p.sommet ;
            p.sommet = p.sommet - 1;
            retourner x ;
    Sinon
            Afficher ('pile vide') ;
Fin
```

7.2 Les Piles

4.4 Files

La file est une structure de données fonctionnant comme la pile. A la différence de la pile, elle applique la stratégie FIFO (First In, First Out). Le premier élément de la file qui est traité en premier.

7.3 Les pointeurs

5.1 Notions de pointeur

Comment fonctionne une variable ?
En réalité une variable est une zone de la mémoire réservée. La mémoire d'un ordinateur est une sorte de casier avec plusieurs compartiments référencés par leurs adresses.
Un pointeur est une adresse mémoire : il permet de désigner directement une zone de la mémoire et donc l'objet dont la valeur est rangée à cet endroit. Un pointeur est souvent typé de manière à préciser quel type d'objet il désigne dans la mémoire.

5.2 Déclaration

On déclare un pointeur en indiquant le type de l'élément pointé, puis on met * avant le nom du pointeur. On peut déjà l'initialiser à null. Dans le programme, il ne faut plus ajouter * au nom du pointeur en l'utilisant

7.3 Les pointeurs

*Type *nomPointeur ;*
nomPointeur = Null ;
L'adresse d'une variable s'obtient en précédant son nom de **&.** Dans

Exemple :

*Prix : **Entier** ;*
***Entier** *p ;*
Prix = 2000
P = &Prix

	Adresse	Valeur
Prix	@408200	2000
P	@632000	@408200

7.3 Les pointeurs

5.3 Propriétés des pointeurs

Un pointeur qui reçoit un autre pointeur pointe alors sur la même variable. Un pointeur précédé de * dans le programme permet d'accéder à la valeur de l'élément pointé.

Exemple

Prix	2000
P	@408200
&Prix	@408200
***P**	2000
&P	@632000

7.3 Les pointeurs

5.4 Arithmétique des pointeurs

La valeur d'un pointeur est un entier. On peut appliquer aux pointeurs quelques opérations arithmétiques.

– Une addition avec un entier ; p + 3
– Une soustraction avec un entier ; p – 3
– Une différence entre deux pointeurs de même type

5.4.1 Addition et soustraction

Soit i un entier, p' = p + 1 (resp. p' = p – 1) désigne un pointeur du même type. p' est alors incrémenté (resp. décrémenté) de la valeur de i multipliée par taille du type du pointeur.
Exemple : Soit &p = @408 de type entier (codé sur 4 octets). p + i = @408 + i*4

7.3 Les pointeurs

5.4.2 Différences entre pointeurs

Soit p et q deux pointeurs sur des éléments du même type. p – q désigne un entier dont la valeur est égale à p – q / taille du type.

FIN DU COURS